Advances in Engineering Software 47 (2012) 62–71

Contents lists available at SciVerse ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

A computational intelligence algorithm for simulation-driven optimization problems

Yoel Tenne

Faculty of Science and Engineering, Kyoto University, Kyoto, Japan

A R T I C L E I N F O

Article history:
Received 9 December 2010
Received in revised form 2 August 2011
Accepted 13 December 2011
Available online 9 January 2012

Keywords:
Expensive functions
Evolutionary algorithms
Modeling
Classification
Model selection
Simulation-driven optimization

A B S T R A C T

The modern engineering design process often relies on computer simulations to evaluate candidate designs. This simulation-driven approach results in what is commonly termed a computationally expensive black-box optimization problem. In practise, there will often exist candidate designs which cause the simulation to fail. Such simulation failures can consume a large portion of the allotted computational resources, and thus can lead to search stagnation and a poor final solution. To address this issue, this study proposes a new computational intelligence optimization algorithm which combines a model and a *k*-NN classifier. The latter predicts which solutions are expected to cause the simulation to fail, and its prediction is incorporated with the model prediction to bias the search towards valid solutions, namely, for which the simulation is expected to succeed. A main contribution of this study is that to further improve the search efficacy, the proposed algorithm leverages on model-selection theory and continuously calibrates the classifier during the search. An extensive performance analysis using an engineering application of airfoil shape optimization shows the efficacy of the proposed algorithm.

1. Introduction

Nowadays, engineers often use *computer simulations* to evaluate candidate designs. Such simulations, which must be properly validated with real-world laboratory experiments, can reduce the time and cost of the design process, and hence have established themselves as a valuable tool. This *simulation-driven* setup entails an optimization problem with three distinct challenges [1]:

- The simulation is often available only as a legacy or a commercial software, whose internal workings are inaccessible to the user. Accordingly, it is treated as a *black-box function* with no analytic expression defining how candidate solutions are mapped to their objective values.
- Each simulation run is *computationally expensive*, that is, requiring a lengthy execution time, and so only a small number of such evaluations can be made during the entire optimization search. and
- The resultant objective function often has a complicated, nonconvex landscape, which makes it difficult to locate an optimum.

An established strategy for dealing such optimization problems is to couple a computational intelligence (CI) optimizer, such as an evolutionary algorithm (EA), particle swarm optimizer (PSO) and alike, with a *model*, namely, a mathematical approximations of the objective function which is computationally cheaper to evaluate. This combination has proven to be effective since the CI optimizer typically performs well on nonconvex objective landscapes, while the model replaces the expensive function and economically provides its predicted objective values to the optimizer.

However, simulation-driven problems introduce another difficulty: the simulation may 'crash' and fail to return an objective value for some candidate designs. We refer to such designs as *simulator-infeasible* (SI), while those for which the simulation completes successfully and provides the output value are termed *simulator-feasible* (SF). SI designs (or vectors) have two main implications for the optimization problem: (a) they do not have an objective value associated with them, which results in a discontinuous objective function and exacerbates the difficulty of the optimization, and (b) such designs can consume a considerable portion of the allotted computational resources without providing new information nor driving the search towards better designs, thus degrading the search effectiveness.

To effectively handle such SI vectors in simulation-driven optimization, we propose a CI algorithm which incorporates a classifier into the optimization search. Specifically, the proposed algorithm retains the SI vectors during the entire search, so they could be used to train the classifier. The classifier is continuously trained using both the SI and SF vectors already evaluated, and its role is to predict if a new vector is SI or not. The proposed algorithm incorporates the classifier's prediction with the model prediction

E-mail address: ytw08-a1@yahoo.com

0965-9978/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.advengsoft.2011.12.009

to bias the search to vectors predicted to be SF. The classifiers used is the k nearest neighbors (k-NN), thanks to its robustness and versatility. However, this classifier requires a user-prescribed parameter, namely, k, the number of nearest neighbors to consider in the classification process. The optimal value of this parameter is unknown a priori, and it maybe impractical to obtain by numerical experiments due to the limited computational budget. Therefore, a main contribution of this paper is the classifier-adaptation stage which continuously selects during the search an optimal k value, to further improve the search efficacy. An extensive performance analysis using an engineering application of airfoil shape optimization shows the efficacy of the proposed algorithm. The remainder of this paper is organized as follows: Section 2 provides the relevant theoretical background on expensive optimization problems, Section 3 describes the proposed algorithm, and Section 4 gives a detailed performance analysis. Lastly, Section 5 summarizes this paper.

2. Theoretical background

2.1. The use of models in expensive optimization problems

Computationally expensive optimization problems are prevalent in engineering and science, and as mentioned in Section 1, a typical scenario is the design optimization process driven by computationally expensive computer simulations. Fig. 1 shows the layout of such problems.

The objective function arising in such problems is often nonconvex, and so classical gradient-based optimizers may converge to a poor local optimum. This has motivated the application of CI optimizers, as they tend to perform well in such challenging objective landscapes. In contrast to classical optimizers which use a single iterate, CI optimizers employ a population of candidate solutions, as well as stochastic operators, and both of these features provide them with enhanced exploration capabilities and better immunity to premature convergence. One such CI optimizer is the evolutionary algorithm (EA), which applies operators inspired by the theory of evolution, namely, selection, recombination and mutation. The fittest solutions survive and propagate through the generations, which results in a gradual adaptation of the population to the function landscape [2].

While CI optimizers do not require gradient information and perform well on nonconvex landscapes, they often require a large number of function evaluations to converge. This poses an obstacle in applying them to expensive optimization problems, where the expensive function can be evaluated only a small number of times. One strategy to overcome this is *modeling*, where a computationally-cheaper mathematical approximation, namely, the model, replaces the true expensive function during most the search. Models are typically interpolants trained with previously evaluated vectors, for example, polynomials, artificial neural networks, radial basis functions (RBF), and Kriging [3]. CI algorithms which use models are commonly termed *model-assisted* or *surrogate-assisted*, and following their established effectiveness [4,5] we adapt the modeling approach in this study.

The high evaluation cost implies only a small number of vectors will be available to train the model, and so it will be inherently

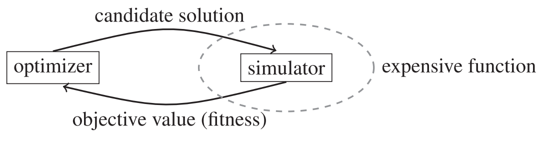

Fig. 1. Principle schematic of simulation-driven optimization problems.

inaccurate. The extent of the model's inaccuracy is unknown a priori, and depends on problem features such as the number of variables, and the complexity of the objective function landscape [6]. If the model inaccuracy is severe, the optimizer may even converge to a false optimum, namely, an optimum of the model which is not an optimum of the true expensive function [7]. As such, model-assisted algorithms must manage the model to ensure convergence to an optimum of the true expensive function.

One established approach for model management is the *trust-region* (TR) framework, which originated in the domain of nonlinear programming. Here, the model is assumed to be accurate in the TR (designated as $\mathscr{T}$), that is, a neighborhood around the current iterate. Starting from an initial guess $\mathbf{x}^{(0)}$, at each iteration, $i = 0, 1, \ldots$, a model is trained and the optimizer performs a *trial step* where it seeks an optimum of the model $m(\mathbf{x})$ in the TR, where

$$\mathscr{T} = \{\mathbf{x} : \|\mathbf{x} - \mathbf{x}^{(i)}\|_2 \leqslant \Delta\}, \tag{1}$$

and Δ is the TR radius. Assuming a minimization problem, this defines the following constrained optimization problem

$$\begin{aligned} \min \quad & m(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{x} \in \mathscr{T} \end{aligned} \tag{2}$$

which yields $\mathbf{x}_{\mathrm{m}}$, the optimum of the model in the TR. The success of the trial step is then measured by the merit value

$$\rho = \frac{f(\mathbf{x}^{(i)}) - f(\mathbf{x}_{\mathrm{m}})}{m(\mathbf{x}^{(i)}) - m(\mathbf{x}_{\mathrm{m}})}, \tag{3}$$

where $\rho > 0$ indicates the trial was successful. Based on the value of ρ the algorithm then updates the iterate and the TR, and different TR updating schemes have been proposed. For example, Wujek and Renaud [8,9] proposed using three ranges of values, where the TR was contracted if ρ was deemed low, enlarged if ρ was deemed high, and no change was made for intermediate values. Another scheme was proposed by Conn et al. [10], and used two threshold values for updating the TR. In this study, we apply the following updating scheme:

- If $\rho > 0$: the trust-region (TR) is centered at $\mathbf{x}_{\mathrm{m}}$ (so $\mathbf{x}^{(i+1)} = \mathbf{x}_{\mathrm{m}}$) and Δ is increased.
- Otherwise: Δ is decreased.

A strong merit of the TR framework is that it guarantees asymptotic convergence to an optimum of the true expensive objective function [11], and this has motivated its use with CI optimizers, for example [5,12–14].

2.2. Simulator-infeasible vectors

As mentioned in Section 1, this study focuses on expensive optimization problems with candidate solutions which cause the simulation code to fail, namely, SI vectors. The difficulties introduced by such vectors are well documented in literature, for example Koehler and Owen [15] mentioned 'inputs combinations which are likely to crash the simulator', Rasheed et al. [16] described a multidisciplinary optimization problem with 'unevaluable points' which 'cause the simulator to crash', while Conn et al. [17] mentioned 'virtual constraints' where 'function evaluations fail at certain points'. Additional studies mentioning this issue include [7,18–22].

Regarding techniques to handle SI vectors, Rasheed et al. [16] used an EA as the optimizer, and proposed using a classifier to screen vectors before evaluating them. Those predicted to be SI were assigned a 'death penalty', that is, a fictitious and highly penalized objective value, to quickly eliminate them from the population. The study did not consider using models, and the EA